Week-9.1

2303A51018

Batch-28

Prompt

Problem 1:

Consider the following Python function:

```
def find_max(numbers):
return max(numbers)
```

Task:

• Write documentation for the function in all three formats:

(a) Docstring

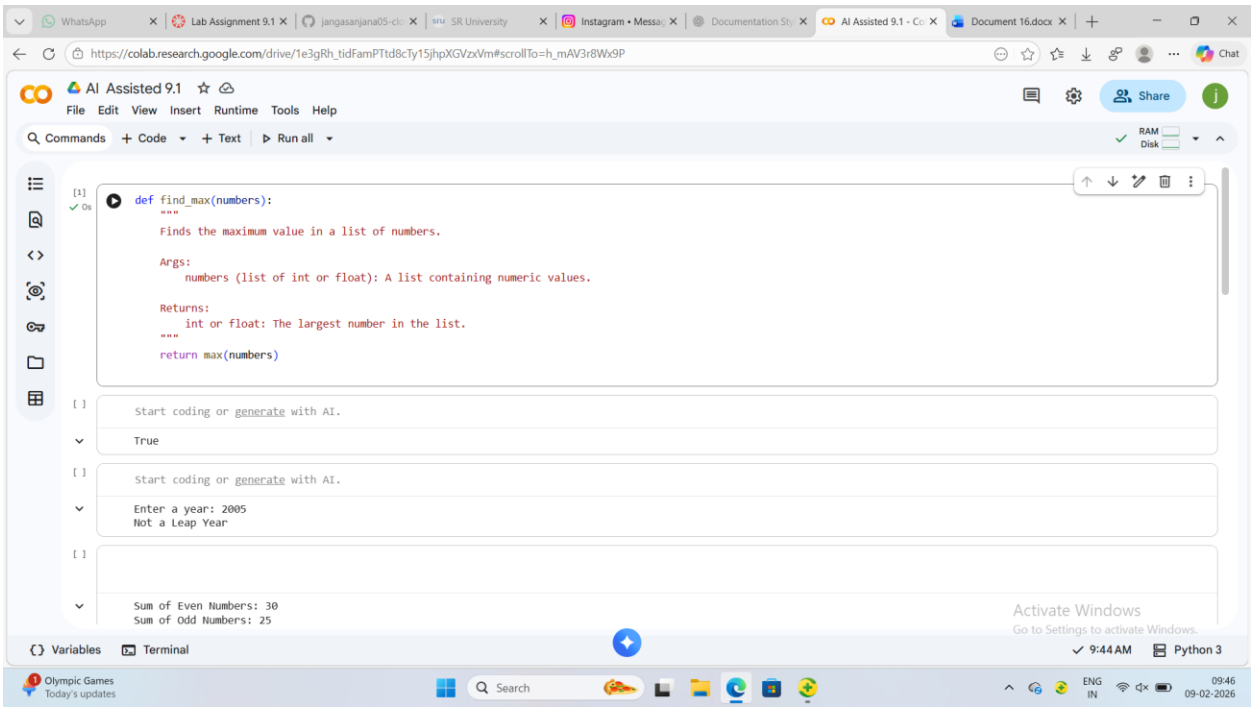
(b) Inline comments

(c) Google-style documentation

• Critically compare the three approaches. Discuss the advantages, disadvantages, and suitable use cases of each style.

• Recommend which documentation style is most effective for a mathematical utilities library and justify your answer.

Code



Critical Comparison of Documentation Styles

Aspect	Docstring (Basic)	Inline Comments	Google-Style Documentation
Purpose	Describes function behavior	Explains specific lines	Provides structured API documentation
Readability	Moderate	High for small logic	Very high
Detail Level	Medium	Low–Medium	High
Tool Support	Limited	None	Excellent (Sphinx, IDEs)
Scalability	Okay for small projects	Poor for large codebases	Excellent
Maintenance	Easy	Can become cluttered	Easy and consistent

Advantages & Disadvantages

1. Docstring (Basic)

Advantages

- Simple and quick to write
- Accessible via `help()` in Python
- Good for small functions

Disadvantages

- No strict structure
- Less suitable for large teams

Use Case

- Small scripts
- Academic examples
- Beginner-level projects

2. Inline Comments

Advantages

- Explains code logic clearly
- Useful for complex steps

Disadvantages

- Can clutter code
- Not useful for API documentation
- Redundant for self-explanatory code

Use Case

- Explaining complex algorithms
- Non-obvious logic

3. Google-Style Documentation

Advantages

- Clear and standardized
- Excellent for collaboration
- Supported by documentation generators
- Very readable and professional

Disadvantages

- Slightly more verbose
- Requires discipline to maintain format

Use Case

- Libraries
- APIs
- Professional and production code

Recommendation for a Mathematical Utilities Library ☑

Recommended Style: Google-Style Documentation

Justification:

- Mathematical utility libraries are reused by many developers
- Clear parameter and return descriptions are critical
- Google-style docstrings are:
    - Easy to read
    - IDE-friendly
    - Automatically convertible into API documentation
- Scales well as the library grows

👉 **Conclusion:**
For a mathematical utilities library, **Google-style documentation is the most effective** because it balances clarity, structure, and professional standards.

Problem 2: Consider the following Python function:

Prompt

def login(user, password, credentials):

return credentials.get(user) == password

Task:

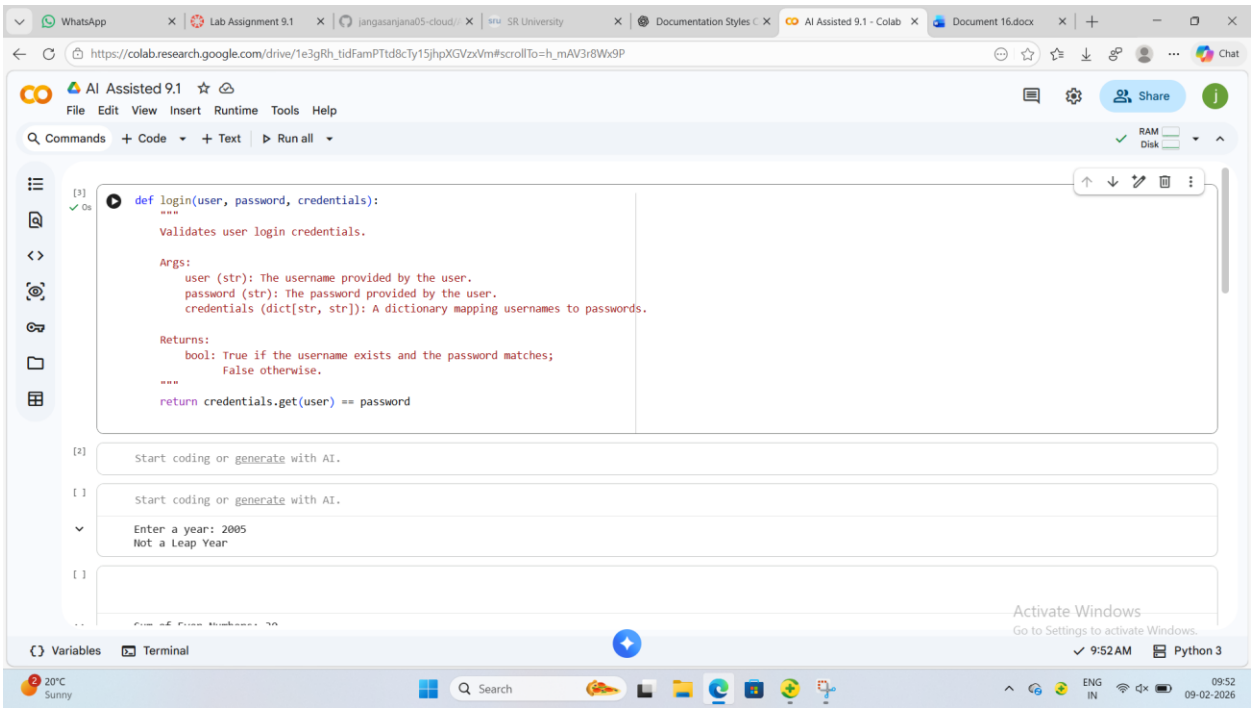
1. Write documentation in all three formats.

2. Critically compare the approaches.

3. Recommend which style would be most helpful for new

developers onboarding a project, and justify your choice.

Code



Critical Comparison of Documentation Approaches

Criteria	Docstring	Inline Comments	Google-Style Documentation
Clarity	Good	Moderate	Excellent
Structure	Loose	None	Highly structured
Beginner Friendly	Yes	Yes (for logic)	Yes
Scalability	Limited	Poor	Excellent

IDE & Tool Support	Basic	None	Strong
Maintenance	Easy	Can get messy	Easy and consistent

Advantages, Disadvantages & Use Cases

Docstring

Advantages

- Simple and readable
- Good overview of function purpose
- Built-in Python support (`help()`)

Disadvantages

- No enforced format
- Less detailed for complex systems

Best Use

- Small projects
- Educational code
- Simple functions

Inline Comments

Advantages

- Explains logic step-by-step
- Helpful for beginners reading code

Disadvantages

- Clutters short or simple functions
- Not useful for API-level understanding

Best Use

- Complex conditions
- Non-obvious security or business logic

Google-Style Documentation

Advantages

- Clear explanation of inputs and outputs
- Industry-standard format
- Easy for teams to follow
- Works well with documentation tools

Disadvantages

- Slightly verbose
- Requires consistency

Best Use

- Team projects
- APIs and libraries
- Long-term maintenance

3. Recommendation for New Developer Onboarding ☑

Recommended Style: Google-Style Documentation

Justification:

- New developers need:
    - Clear purpose of the function
    - Exact meaning of parameters
    - Expected return values

- Google-style documentation:
    - Reduces confusion
    - Improves readability
    - Helps developers understand code without digging into implementation
- IDEs can display this documentation instantly during coding

👉 **Conclusion:**
For onboarding new developers, **Google-style documentation is the most helpful** because it is structured, beginner-friendly, and professional, making it easier to understand both *what* the function does and *how* to use it.

Problem 3: Calculator (Automatic Documentation Generation)

Task: Design a Python module named calculator.py and

demonstrate automatic documentation generation.

Instructions:

1. Create a Python module calculator.py that includes the

following functions, each written with appropriate docstrings:

o add(a, b) – returns the sum of two numbers

o subtract(a, b) – returns the difference of two numbers

o multiply(a, b) – returns the product of two numbers

o divide(a, b) – returns the quotient of two numbers

2. Display the module documentation in the terminal using

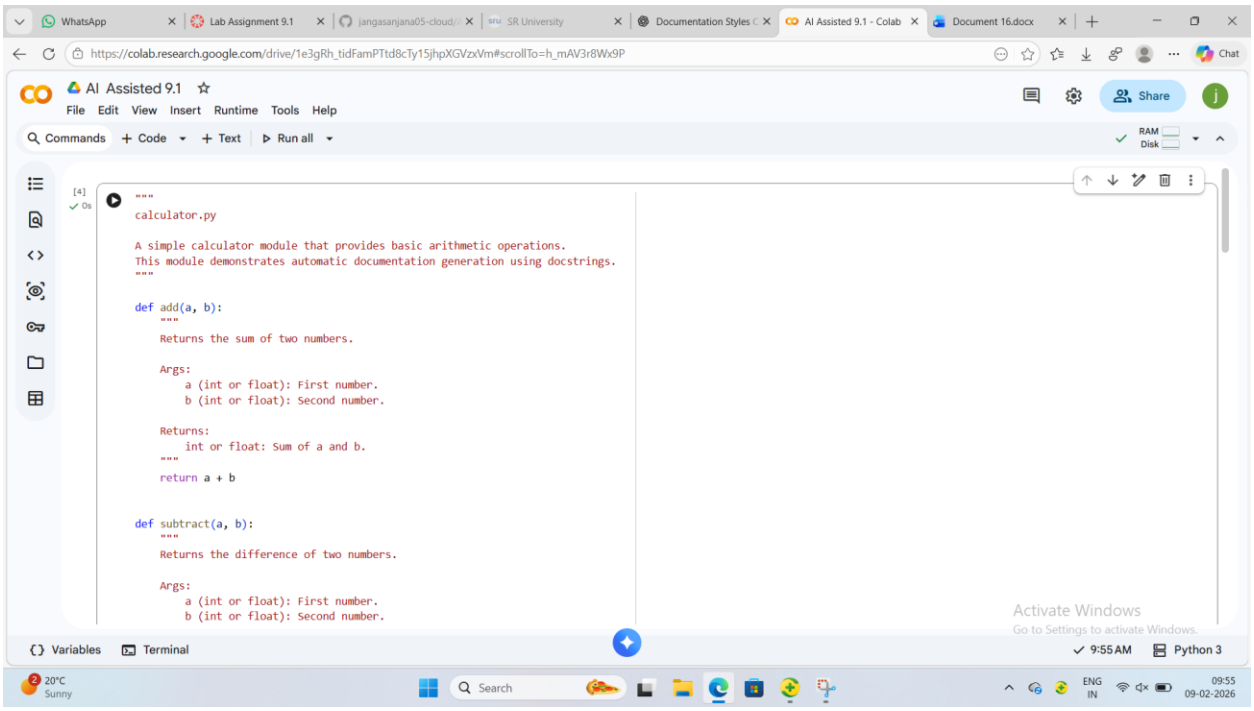
Python's documentation tools.

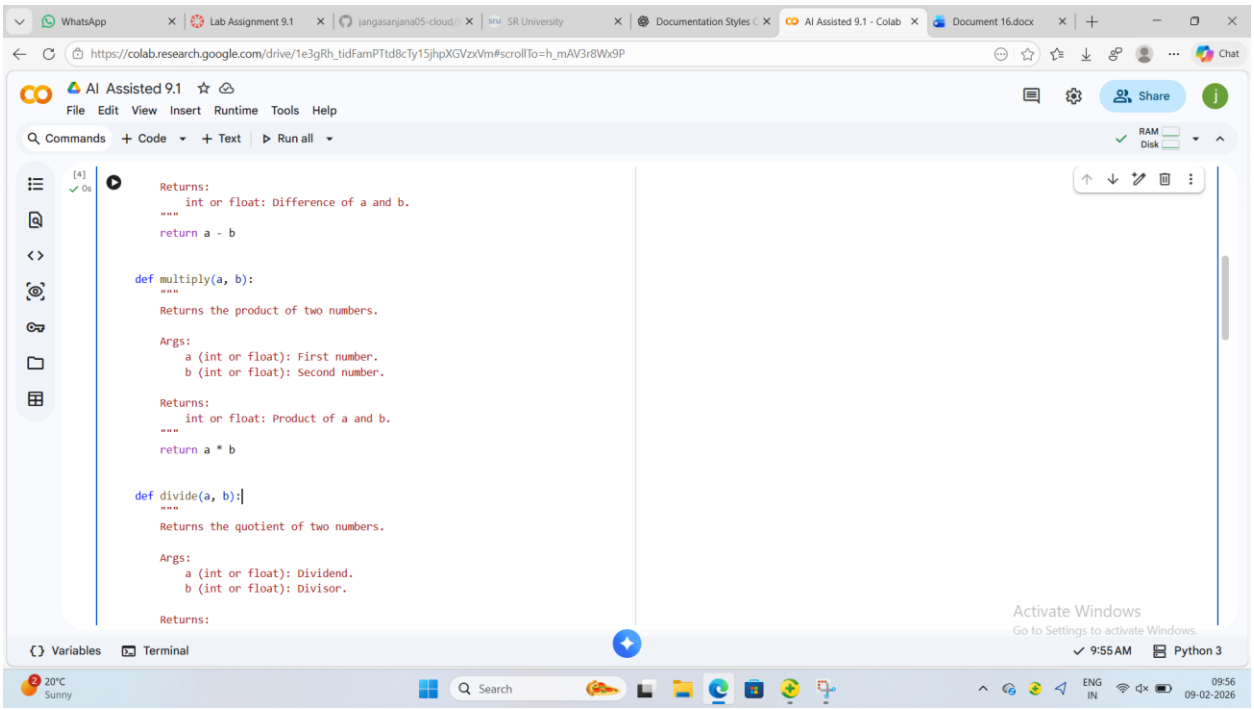
3. Generate and export the module documentation in HTML

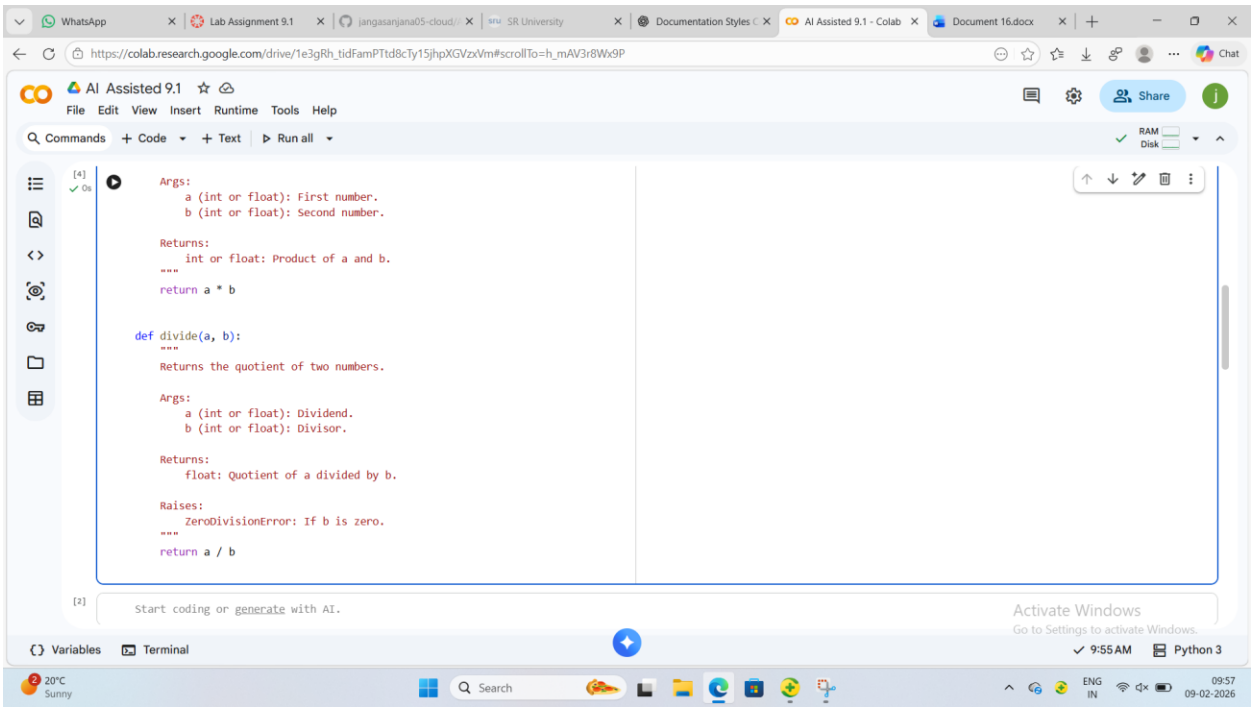
format using the pydoc utility, and open the generated HTML

file in a web browser to verify the output.

Code


AI Assisted 9.1
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
[4]
0s
"""
calculator.py
A simple calculator module that provides basic arithmetic operations.
This module demonstrates automatic documentation generation using docstrings.
"""
def add(a, b):
"""
Returns the sum of two numbers.
Args:
a (int or float): First number.
b (int or float): Second number.
Returns:
int or float: Sum of a and b.
"""
return a + b
def subtract(a, b):
"""
Returns the difference of two numbers.
Args:
a (int or float): First number.
b (int or float): Second number.
Activate Windows
Go to Settings to activate Windows.
Variables
Terminal
9:55 AM
Python 3


AI Assisted 9.1
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
[4]
0s
Returns:
int or float: Difference of a and b.
"""
return a - b
def multiply(a, b):
"""
Returns the product of two numbers.
Args:
a (int or float): First number.
b (int or float): Second number.
Returns:
int or float: Product of a and b.
"""
return a * b
def divide(a, b):
"""
Returns the quotient of two numbers.
Args:
a (int or float): Dividend.
b (int or float): Divisor.
Returns:
Activate Windows
Go to Settings to activate Windows.
Variables
Terminal
9:55 AM
Python 3



2. Display the module documentation in the terminal

Method 1: Using `help()` in Python

Open the terminal in the folder containing `calculator.py` and start Python:

```
python
```

Then run:

```
import calculator
help(calculator)
```

📌 Output will show:

- Module description
- List of functions
- Docstrings for each function

This proves **automatic documentation generation via docstrings**.

Method 2: Using pydoc in terminal (text format)

```
pydoc calculator
```

This displays formatted documentation directly in the terminal.

3. Generate HTML documentation using pydoc

Step 1: Generate HTML file

Run this command in the same directory as `calculator.py`:

```
pydoc -w calculator
```

✅ This generates a file named:

```
calculator.html
```

Step 2: Open the HTML file in a web browser

On Windows:

```
start calculator.html
```

On macOS:

```
open calculator.html
```

On Linux:

```
xdg-open calculator.html
```

📌 The browser will show:

- Module description
- Function list
- Parameters, return values, and exceptions
  (all extracted automatically from docstrings)

Explanation: Automatic Documentation Generation

- Python **docstrings** act as the single source of truth
- Tools like:
  - `help()`
  - `pydoc`
  - IDEs (VS Code, PyCharm)
- automatically extract and format documentation
- No extra documentation files are needed

Conclusion (Exam-Friendly)

- The `calculator.py` module demonstrates **automatic documentation generation**
- Docstrings enable:
  - Terminal-based documentation (`help`, `pydoc`)
  - Web-based HTML documentation (`pydoc -w`)
- This approach improves:
  - Code readability
  - Maintainability
  - Developer onboarding

Problem 4: Conversion Utilities Module

Task:

1. Write a module named conversion.py with functions:

o decimal_to_binary(n)

o binary_to_decimal(b)

o decimal_to_hexadecimal(n)

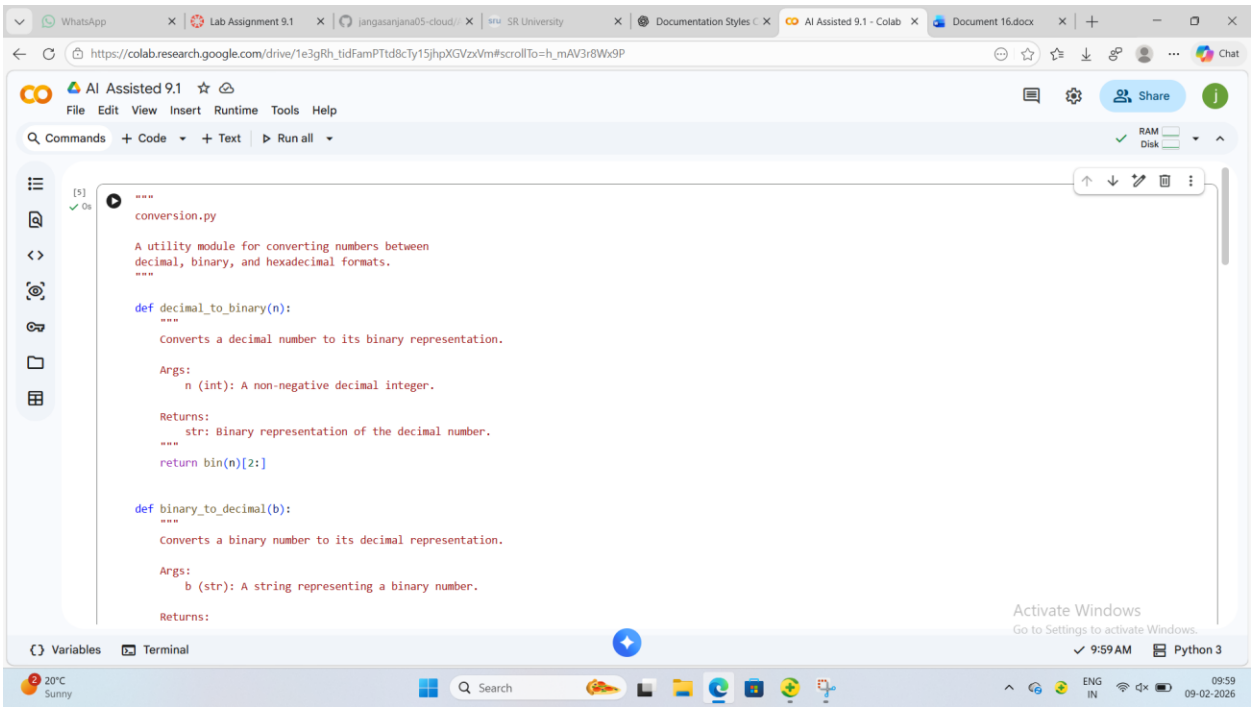
2. Use Copilot for auto-generating docstrings.

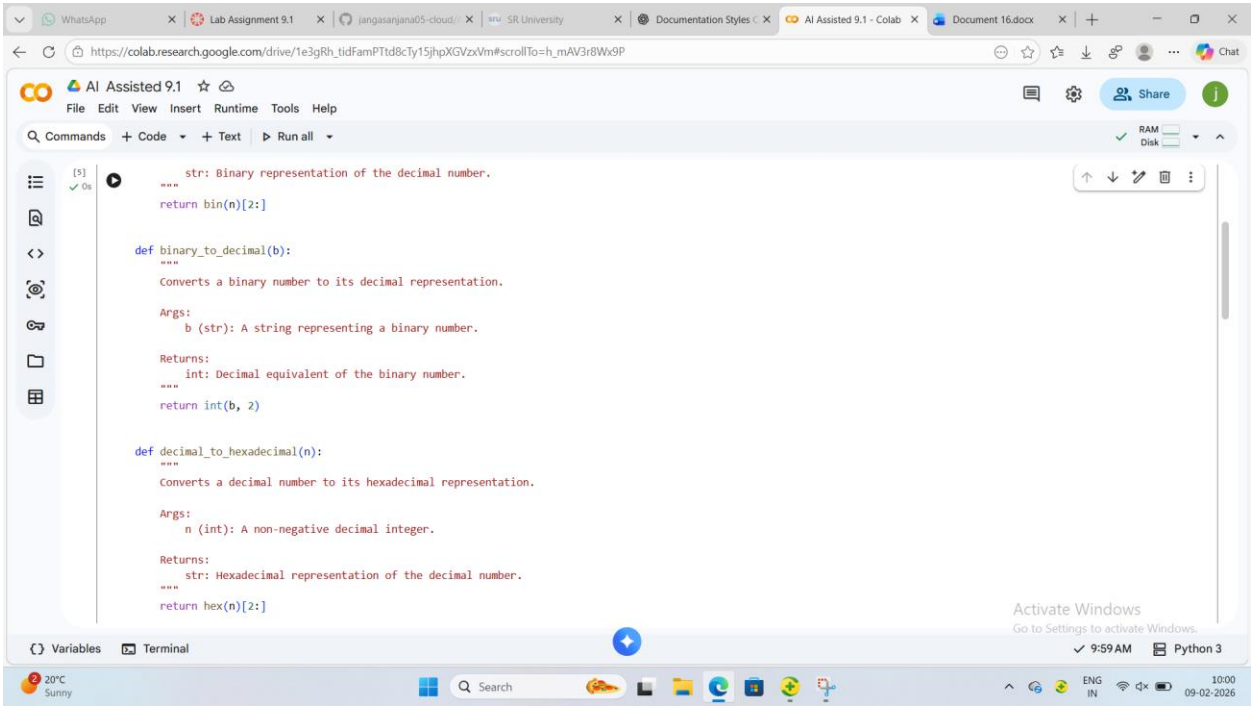
3. Generate documentation in the terminal.

4. Export the documentation in HTML format and open it in a

Browser.

Code





2. Using Copilot for auto-generating docstrings (Explanation)

When using **GitHub Copilot**:

1. Write the function signature (e.g., `def decimal_to_binary(n):`)

2. Press **Enter**
3. Copilot automatically suggests a complete docstring
4. Accept the suggestion using **Tab**

📌 Copilot analyzes:

- Function name
- Parameters
- Return type

and generates meaningful documentation automatically.

3. Generate documentation in the terminal

Method 1: Using `help()`

Open the terminal in the directory containing `conversion.py`:

```
python
```

Then run:

```
import conversion
help(conversion)
```

📌 This displays:

- Module description
- List of functions
- Docstrings for each function

This confirms **automatic documentation generation from docstrings**.

Method 2: Using pydoc (terminal view)

```
pydoc conversion
```

This prints formatted documentation directly in the terminal.

4. Export documentation in HTML format and open in browser

Step 1: Generate HTML documentation

Run:

```
pydoc -w conversion
```

✅ This creates a file named:

```
conversion.html
```

Step 2: Open the HTML file

On Windows:

```
start conversion.html
```

On macOS:

```
open conversion.html
```

On Linux:

```
xdg-open conversion.html
```

📌 The browser will display:

- Module overview
- Function descriptions
- Parameters and return values
  (all extracted automatically from Copilot-generated docstrings)

Conclusion (Exam / Lab Record Ready)

- The `conversion.py` module demonstrates **automatic documentation generation**
- GitHub Copilot assists by auto-generating accurate docstrings
- Python tools like:
  - `help()`
  - `pydoc`
  - `pydoc -w`

Problem 5 – Course Management Module

Task:

1. Create a module course.py with functions:

o add_course(course_id, name, credits)

o remove_course(course_id)

o get_course(course_id)

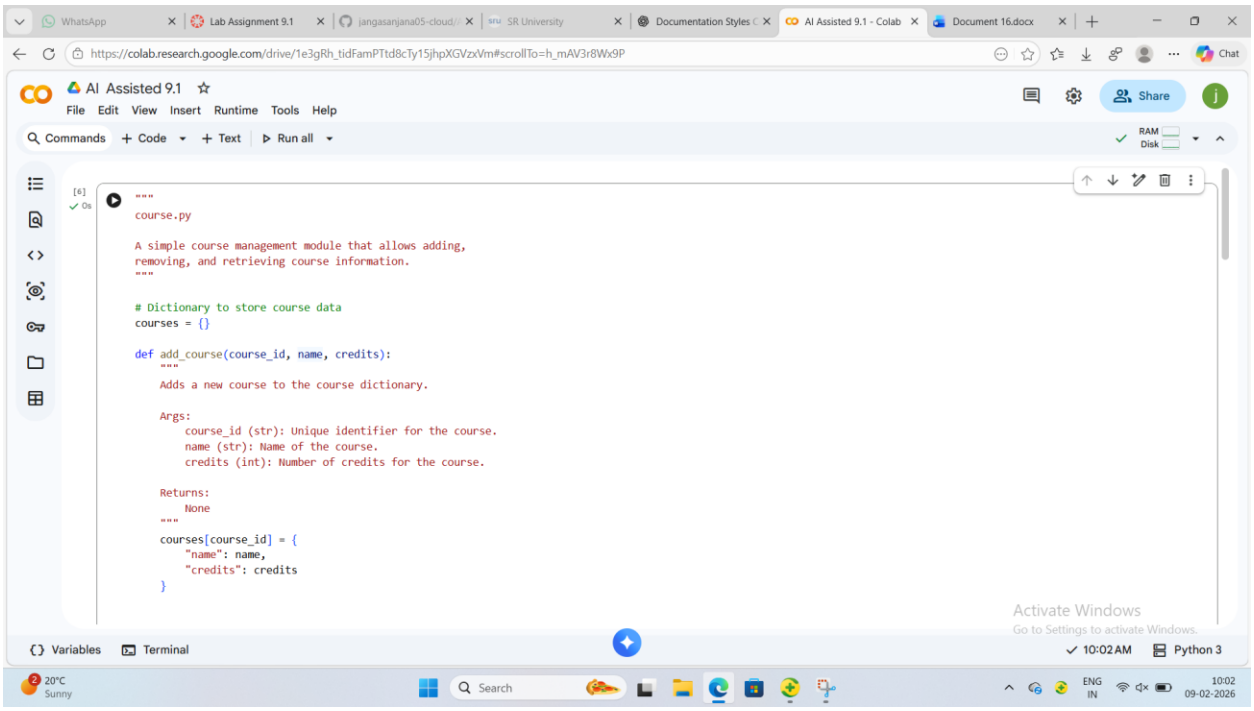
2. Add docstrings with Copilot.

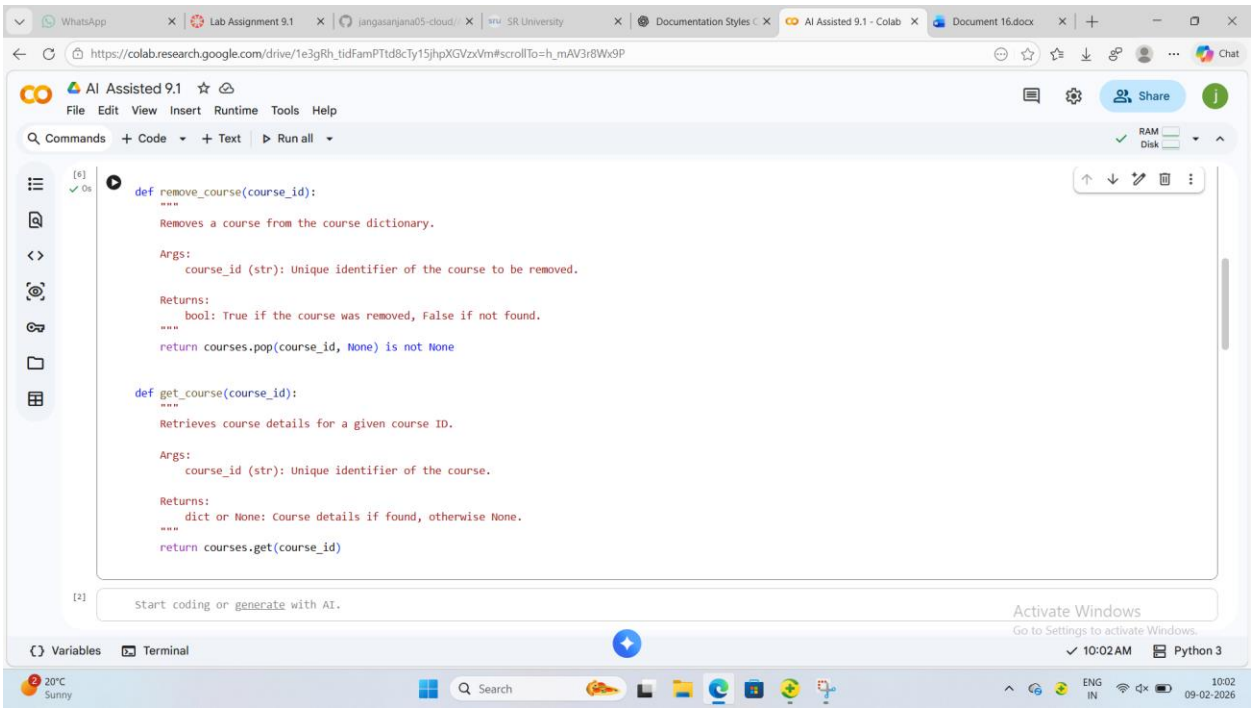
3. Generate documentation in the terminal.

4. Export the documentation in HTML format and open it in a

browser.

Code





2. Adding docstrings with Copilot (Explanation)

Using **GitHub Copilot**:

1. Type the function definition
2. Press **Enter**

3. Copilot suggests a full docstring automatically
4. Accept it using **Tab**

📌 Copilot analyzes:

- Function name
- Parameters
- Return values

and generates meaningful documentation instantly.

3. Generate documentation in the terminal

Method 1: Using `help()`

Open terminal in the folder containing `course.py`:

```
python
```

Then:

```
import course
help(course)
```

📌 Output includes:

- Module description
- Global variables
- Function documentation (from docstrings)

Method 2: Using pydoc (terminal format)

```
pydoc course
```

This displays structured documentation directly in the terminal.

4. Export documentation in HTML format and open in a browser

Step 1: Generate HTML documentation

```
pydoc -w course
```

✅ This creates:

```
course.html
```

Step 2: Open the HTML file

On Windows:

```
start course.html
```

On macOS:

```
open course.html
```

On Linux:

```
xdg-open course.html
```

📌 The browser shows:

- Module overview
- Course management functions

- Parameters and return values
  (all auto-generated from docstrings)

Conclusion (Exam / Lab Record Friendly)

- The `course.py` module demonstrates **automatic documentation generation**
- Copilot assists in writing accurate docstrings
- Python tools:
  - `help()`
  - `pydoc`
  - `pydoc -w`
- automatically generate **terminal and HTML documentation**

✅ Benefits:

- Faster development
- Cleaner documentation
- Easier onboarding for new developers